Learning to Fly: Computational Controller Design for Hybrid UAVs with Reinforcement Learning

by

Jie Xu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author . . . .
Department of Electrical Engineering and Computer Science
May 16, 2019

Certified by . . . .
Wojciech Matusik
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Learning to Fly: Computational Controller Design for Hybrid UAVs with Reinforcement Learning

by

Jie Xu

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Hybrid unmanned aerial vehicles (UAV) combine advantages of multicopters and fixed-wing planes: vertical take-off, landing, and low energy use. However, hybrid UAVs are rarely used because controller design is challenging due to its complex, mixed dynamics. In this work, we propose a method to automate this design process by training a mode-free, model-agnostic neural network controller for hybrid UAVs. We present a neural network controller design with a novel error convolution input trained by reinforcement learning. Our controller exhibits two key features: First, it does not distinguish among flying modes, and the same controller structure can be used for copters with various dynamics. Second, our controller works for real models without any additional parameter tuning process, closing the gap between virtual simulation and real fabrication. We demonstrate the efficacy of the proposed controller both in simulation and in our custom-built hybrid UAVs. The experiments show that the controller is robust to exploit the complex dynamics when both rotors and wings are active in flight tests.

Thesis Supervisor: Wojciech Matusik
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my advisor Prof. Wojciech Matusik, for his innovative ideas, generous support and encouragement in my research. Without his insightful guidance and encouragement, I would never try such a challenging project where I had entirely no experience before and make it happen.

I would like to thank all my collaborators, Tao Du, Adriana Schulz, Bo Zhu, Michael Foshey, Beichen Li. It is my honor to work with so many top researchers. Without their help and commitment to the project, this work would be never possible.

I would also like to thanks all my friends, who play sports, dine out and hold activities with me. They are super important to me to keep my study-life balance.

Finally, I would like to give a big thank to my family, for their long-term cultivation and never-ending support. They are always behind me whenever I make progress or get stuck in my life or study. Their love is my most precious treasure.

Contents

List of Figures

List of Tables

Chapter 1

Intoduction

Multicopters are becoming increasingly popular due to their flight flexibility and stable hovering capability. Fixed-wing airplanes, on the other hand, are more energy efficient during level flight, making them better vehicles for long distance flights. In order to leverage the advantages of both designs, new hybrid UAVs which equip multicopters with a pair of fixed wings have piqued the interest of the aircraft community [20, 31, 7, 27]. These vehicles can quickly switch between hover and flight modes, allowing for highly stable, energy efficient flights. There are many ways to combine multicopters and fixed wings, resulting in hybrid UAVs with vastly different configurations.

Controlling such hybrid designs, however, is difficult because the aerodynamics changes dramatically during the flight. In typical designs, experts manually design controllers for both the *copter flight mode* and the *plane flight mode*, as well as controllers for transitioning between these modes. Designing controllers for the transition phase is especially challenging due to the complexity of the dynamics when rotors and wings are both active. Furthermore, due to the vast differences between various configurations of hybrid UAVs, controllers cannot be easily transferred between designs. In order to construct a new hybrid UAV, engineers have to redesign the controller architecture and its parameters. This makes the whole process extremely labor-intensive and therefore only a small fraction of the possible hybrid UAVs design space has been explored so far.

To address these challenges, we propose a method that automatically computes a neural network controller for a hybrid UAV. Compared with the traditional mode-based, model-specific controllers, our *unified* controller has the following immediate benefits: first, our mode-free, velocity-tracking controller works directly with the full dynamics during the flight. Our controller does not need to differentiate between the copter and flight modes or explicitly deal with the transition between modes. For example, the controller will automatically orient a tail-sitter hybrid UAV purely based on the input velocity – it will set it to a copter orientation for lower velocities and a plane orientation for higher velocities. Second, the architecture of our model-agnostic neural network controller can be used for almost any hybrid UAV configuration. In the future, this could allow knowledge acquired in training one hybrid UAV model to be directly transferred to another design's controller.

A natural choice to train neural network controllers is using reinforcement learning, since these algorithms have been successfully applied on models with complex dynamics [35, 34, 19, 41]. However, while this approach can be easily applied to train hybrid UAVs in a simulation environment, using these controllers for real flights is known to be challenging due to the gap between simulation and reality. To close this gap, we propose a novel error integral block in the state vector for the input of our network, enabling the elimination of steady-state drift errors that are challenging to remove manually. Additionally, we also devise a simulation infrastructure enhanced by noise and latency patterns and a reinforcement learning framework with novel reward functions for hybrid UAVs. We demonstrate the capabilities of our method by fabricating real hybrid UAVs and showing that our controller is robust for real flights.

In summary, our work has the following technical contributions:

- we present an automatic way for designing a *mode-free*, *model-agnostic* neural network controller for real hybrid UAVs.

- we propose a method to improve the robustness of the trained neural network controllers for hybrid UAVs and close the reality gap between simulation and real flight.

Chapter 2

Related Work

2.1 Computational Design for Fabrication

Computational design of complex functional mechanisms is a growing research field. This body of work aims at (1) making it more efficient for experts to explore design spaces and find unprecedented solutions and (2) lowering the barrier of entry for design by casual users, enabling a new age of customization. Recent advances in the field propose new interactive techniques to design mechanical characters [11], robotic creatures [24], creatures with wheels and legs [18], and multicopters [16]. Our method builds upon this research and proposes a tool for designing hybrid UAVs by mixing and matching parts from an expert-designed collection [36, 13].

Computational design tools guide users in creating objects based on how they should function once manufactured. Therefore, the tools need to support *performance-driven* design exploration [39] or design optimization [40, 9]. Performance evaluations are typically a bottleneck for these methods. This process is particularly challenging for cyberphysical systems whose performance depends not only on geometric considerations but also on motion planning and control. In the context of a hybrid UAV, measuring its flight performance requires designing an adequate controller in addition to designing its structure. Until now, the design process for hybrid UAV controllers had to be done manually due to the complexity of system dynamics. Our paper bridges this gap, proposing a new automatic method that can design controllers for a large variety of hybrid UAV designs.

2.2 Hybrid UAVs

Hybrid UAVs have demonstrated outstanding aerodynamic performance and energy saving capabilities. However, they also bring new challenges in controller design. A large variety of dynamic models and control algorithms have been developed to accommodate UAVs with different wing configurations (see e.g. [12]). For example, a simple flat-plate wing model coupled with a TVLQR controller has demonstrated success in enabling precise control of a fixed-wing drone with fast locomotion and varying trajectories [8]. This design supports a single flying mode and multiple predefined trajectories, relying on online search to select the best control outputs from the precomputed data.

Controlling a multi-mode UAV requires a complicated strategy due to the differences between flying modes, which rely on distinct flying mechanisms. A straightforward strategy is to aggregate multiple controllers with each one focused on a single flying mode. These modes can be organized as a state transition graph [20] or a cascading hierarchy [31, 7]. A number of intermediate modes need to be incorporated to enable smooth transitions between different modes. A typical example of this approach is the design of a tail-sitter drone, which takes off vertically, hovers in the air statically, and glides horizontally [27]. Such methods produce reliable control mechanisms but lack applicability to different drone designs. Relatively little attention has been paid to a universal controller that can be applied to different UAV models.

2.3 Reinforcement Learning

Deep reinforcement learning has become a prevalent tool for training continuous controllers for a variety of robots to finish highly dynamic tasks in animation. Some recent successful examples include walkers [19], humanoid robots with acrobatic or athletic skills [30, 28, 23], robotic manipulators [14], and aerial robots [41, 42]. The work most related to ours is [42], which trained a flying dragon actuated by flapping wings to follow a trajectory. Their flying robot modelled aerodynamic effects between wings and airflow, but the method was not tested on hardware platforms.

While reinforcement learning has produced convincing results in animation, successful demonstrations on real robots are rare due to discrepancies between simulated and real environments. A few notable papers have managed to close this gap for specific types of robots through domain randomization [38, 32, 3, 37], more accurate modeling [6, 17], or better sensing [21]. Designing controllers for hybrid UAVs, however, poses new challenges not addressed in the previous work—these vehicles have a floating base, highly dynamic motion, and unsteady aerodynamic effects. Our experiments show that existing techniques are not sufficient for training a robust controller for a real hybrid UAV, which motivates us to propose neural network controllers with new inputs and a novel integral block. Moreover, controllers in most of the prior research worked for a specific robot design, while we present a method that works robustly for a variety of hybrid UAVs with significantly different topology and dynamics.

Our method is also related to [29] in that both propose neural network controllers with time-dependent blocks to capture the mismatch between simulated and real environments. Comparing to the LSTM network controller in their method which uses a memory-based policy in a recurrent block, we encode the history information in a concise integral block, which helps reduce the workload of the training process significantly.

Our work draws inspiration from works on controlling real helicopters using reinforcement learning [22, 2] and learning from demonstrations [10, 1]. All the prior work and ours show controllers on real UAVs; however, the scopes are quite different. In the prior work, a functional controller on hardware is provided for a pilot expert to demonstrate flight patterns, then the method either learns an accurate dynamics model or learns an improved controller from the flight data. Our work is different in that we design controllers directly from the inaccurate dynamics of the hybrid UAVs without needing a real, working controller beforehand.

Chapter 3

System Overview

In this chapter, we briefly describe the overview of our whole computational controller design system.

As shown in Figure 3-1, we propose a computational pipeline to design the controller for hybrid UAVs. Users design the geometry of a hybrid UAV by selecting and matching parts from a component data set. This data set contains parametric components designed by experts and composition rules that constrain how the components can be assembled [16, 36, 18]. We built a numerical simulator to reproduce the real experimental environment for UAVs that takes into account random sensor noise, variances in system identification, and delay in control signals (Section 4).

At the heart of our hybrid UAV control algorithm is a novel and unified neural network controller to support mode-free motion control (Section 5). We aim to tackle two long-standing challenges in the state-of-the-art hybrid UAV control algorithms. First, we want to automate the flying mode transitions for a single aircraft. To this end, we replace all the modes and their discrete transitions, which are usually hard-coded in a hybrid controller, with a mode-free continuous neural network. Second, we aim to automate the transfer from a network trained in a virtual simulator to a controller that can fly a real hybrid UAV. We formulate a new state vector as the input of our network by incorporating an error integral term, enabling a universal solution to eliminate the steady-state drift error that is challenging to fix even manually.

These two essential contributions, in conjunction with our aircraft simulator en-

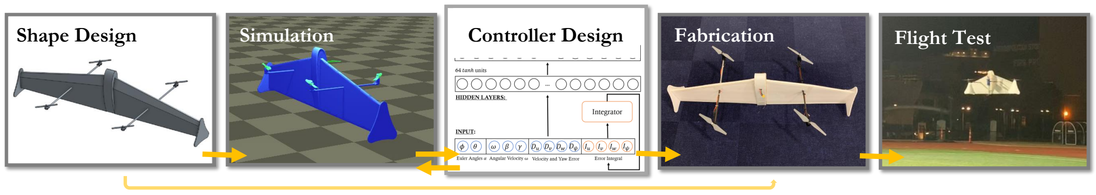

Figure 3-1: In our system, users first design the geometry of a hybrid UAV (left most). The design is then used in our realistic simulator (middle left) to compute the corresponding neural network controller using reinforcement learning (middle). Once the training is done, we construct the corresponding hybrid UAV based on the geometry and controller specifications (middle right). Finally, we verify the UAV performance in real flight tests (right).

hanced with noise perturbations (Section 4) and the reinforcement learning framework with a novel reward function (Section 6), establish a full pipeline to automate the processes of controller design and parameter tuning for a customized hybrid UAV. Finally, we evaluate the efficacy of the neural network controller, state representation, and the fully automated pipeline in numerical settings (Section 7.3). We finish the pipeline by manufacturing the vehicle based on the geometry specifications, set its controller code based on the manufactured vehicle, and conduct real flight experiments (Section 8).

Chapter 4

Hybrid UAVs Simulation

In this chapter, we introduce our physics-based simulation for hybrid UAV, which provides a high-fidelity virtual environment to later train a robust controller in it.

4.1 Dynamic Model

The state variables of the hybrid UAV system are represented by a twelve-dimensional vector in phase space:

$$\mathbf{q} = (x, y, z, \phi, \theta, \psi, u, v, w, \omega, \beta, \gamma)^T \in \mathbf{R}^{12}, \tag{4.1}$$

which can be subdivided into four three-dimensional vectors in sequence. These four state vectors include position $\mathbf{x} = (x, y, z)^T$, the Euler angles (i.e. roll, pitch and yaw) $\boldsymbol{\phi} = (\phi, \theta, \psi)^T$, velocity $\mathbf{u} = (u, v, w)^T$, and angular velocity $\boldsymbol{\omega} = (\omega, \beta, \gamma)^T$.

Following the Newton's second law, the time derivative of the system's linear momentum can be written as

$$m\ddot{\mathbf{x}} = \mathbf{f}^G + \mathbf{f}^T + \mathbf{f}^L + \mathbf{f}^D, \tag{4.2}$$

with m as mass, $\mathbf{f}^G$ as gravity, $\mathbf{f}^T$ as thrust, $\mathbf{f}^L$ as wing lift and $\mathbf{f}^D$ as wing drag.

Following Euler's rotation equations, the time derivative of the system's angular momentum can be written as

$$\mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} = \sum_{i=1}^{N}(\lambda_i T_i \mathbf{d}_i + \mathbf{r}_i \times T_i \mathbf{d}_i) + \sum_{j=1}^{M}[\mathbf{s}_j \times (\mathbf{f}_j^L + \mathbf{f}_j^D)], \tag{4.3}$$

with inertial tensor $\mathbf{J}$, propeller torque-thrust ratio λ, propeller thrust T, propeller direction $\mathbf{d}$, propeller position in body frame $\mathbf{r}$, and the mass center of wing in body frame $\mathbf{s}$. The number N and M denote the number of propellers and wings, respectively.

We follow the flat-plane model [12] to approximate the lift and drag on each wing as:

$$\mathbf{f}^L = \rho|\mathbf{u}|^2 cos(\alpha) sin(\alpha) S \tag{4.4}$$

$$\mathbf{f}^D = \rho|\mathbf{u}|^2 sin^2(\alpha) S \tag{4.5}$$

with ρ as the air density, α as the angle of attack, and S as the surface area of each wing. The aerodynamic interactions among the propellers, wings, and fuselage are ignored in our dynamic model.

4.2 Numerical Simulation

We build a rigid-body simulator incorporating this simplified aerodynamic model to support the controller training process in a virtual environment. Our simulator mimics a closed feedback loop. In particular, the controller receives its input from both the pilot and the sensor and generates the control signals which are then used to update the rotor thrusts at each time step. The phase variables are updated with an explicit Euler time integration using Equations 4.2 to 4.5. We also model noise in a real feedback control loop by applying different numerical perturbations in the system such as adding noise to sensor readings, perturbing mass and inertia, adding noise to lift and drag, and perturbing the latency of control signals and the time step length of the control loop.

Chapter 5

Neural Network Controller for Hybrid UAVs

We design a neural network controller that is mode-free and model-agnostic to automate the control parameter tuning processes for various customized UAV designs. In particular, we incorporate an error integral term in our state vector to enable the transfer from a computationally trained model to a real copter.

5.1 Network Architecture

The architecture of our novel neural network controller is illustrated in Figure 5-1. We design a neural network with two fully-connected hidden layers. Each layer is composed of 64 *tanh* units. The input state $\mathbf{s} \in \mathbf{R}^{13}$ is represented by a vector with fixed dimensions and the output of the network is an N-dimensional vector $\mathbf{a}$ specifying the control signal for each rotor, with N as the total number of rotors equipped on the aircraft.

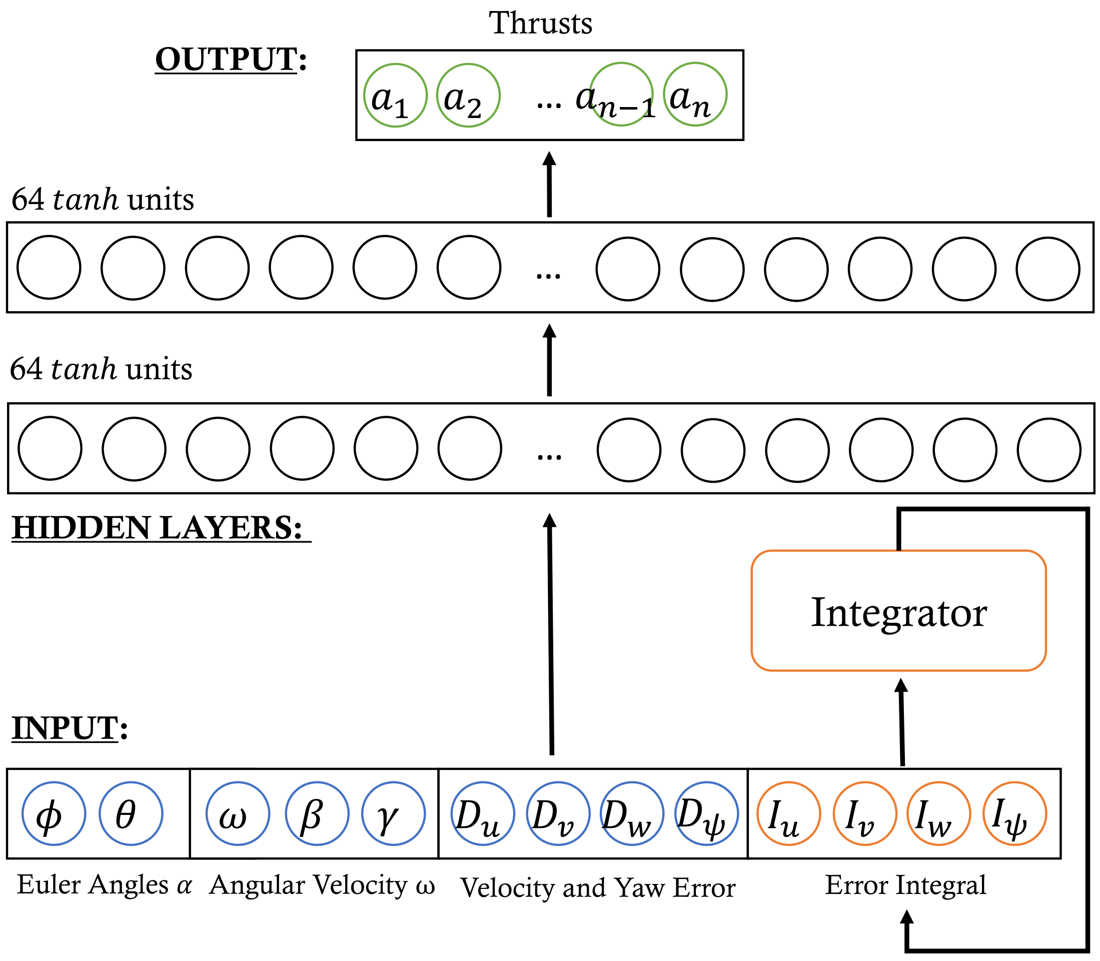

Figure 5-1: The neural network controller takes a thirteen-dimensional vector consisting of angle, angular velocity, error, and its integral as input. The neural network consists of two layers with 64 *tanh* units each and a linear layer to output the final thrust for each rotor.

5.2 State Variables

As illustrated in Figure 5-1, we define the input state vector of a hybrid UAV for our neural network controller as:

$$\mathbf{s} = (\phi, \theta, \omega, \beta, \gamma, D_u, D_v, D_w, D_\psi, I_u, I_v, I_w, I_\psi)^T \in \mathbf{R}^{13}, \tag{5.1}$$

where D_u,D_v,D_w are the differences between the target and the current velocity components in a local coordinate system, D_ψ is the difference between the target and the current yaw angle, and I_u, I_v, I_w, I_ψ are the convolutional integrals of D_u,D_v,D_w, and D_ψ, respectively. Notice that each component of $\mathbf{s}$ is a function of time t. We omit the symbol t in the following context for conciseness.

Compared to a standard state vector of a hybrid UAV (such as $\mathbf{q}$ in Equation 4.1), we design our network input vector following these philosophies: first, we remove position from the state because we track velocity only for UAV motion control. We have chosen to use a velocity-tracking controller not only because it works as a unified control representation for various types of hybrid UAVs where an on-board positioning system is not always available, but also because it is a more intuitive framework for the pilot. Second, we record the velocity difference instead of recording both vectors to reduce the dimension of the space. Third, we append a four-dimensional integral term to the end to compensate for accumulated error terms regarding both velocity and yaw angle (see Section 5.3).

If we further define $\bar{\boldsymbol{\phi}} = (\phi, \theta)^T$, $\mathbf{D} = (D_u, D_v, D_w, D_\psi)^T$, and $\mathbf{I} = (I_u, I_v, I_w, I_\psi)^T$, we can rewrite $\mathbf{s}$ in a blockwise manner as

$$\mathbf{s} = (\bar{\boldsymbol{\phi}}^T, \boldsymbol{\omega}^T, \mathbf{D}^T, \mathbf{I}^T)^T, \tag{5.2}$$

where now the state vector is a combination of the roll and pitch angles, the angular velocity, an error vector consisting of linear velocity and yaw, and the time integral of the error vector.

Finally, we give the definition of $\mathbf{D}$ as the difference between the target state vector and the current state vector in a local frame specified by the current yaw angle:

$$\mathbf{D} = \begin{bmatrix} \mathbf{R}_z(-\psi) & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\psi} \end{bmatrix} - \begin{bmatrix} \mathbf{u} \\ \psi \end{bmatrix} \right). \tag{5.3}$$

The first three components of the target vector are the target velocity $\hat{\mathbf{u}}$ and the last component is the target yaw angle $\hat{\psi}$. Both the target velocity and the target yaw are in world space. Similarly, the current state vector is composed of the current velocity and the current yaw angle in world space. The difference between these two vectors are then transformed into a local space by a homogeneous matrix specified by a rotation with the angle of $-\psi$ along the world-down axis (we use North-East-Down coordinates for both world and body frames). We want to highlight that this rigid transformation always aligns the motion in the forward and right directions with the components of D_u and D_v, which is the key to enabling an efficient learning process for the network.

5.3 Error Integral

A distinguishing feature of our state vector compared with previous work is the introduction of the temporally accumulated errors of linear velocity and yaw angle. We will later show that this error term is the critical factor that enables effective transfer of control policies from one aircraft configuration to another (see Section 7.3).

We define the error integral term I as the convolution of $\mathbf{D}$ and a smoothing kernel δ:

$$\mathbf{I}(t) = \int_0^t \mathbf{D}(\tau)\delta(t-\tau)d\tau, \tag{5.4}$$

The discrete form of Equation 5.4 can be obtained by discretizing the time axis with n time steps as

$$\mathbf{I}(n) = \sum_{i=1}^{n} \mathbf{D}(i)\eta^{n-i}, \tag{5.5}$$

in which a constant decay coefficient η is used to incrementally attenuate the impact of the errors in previous time steps, as well as to avoid numerical explosion after a long time interval.

We can further obtain a recursive expression of Equation 5.5 to update $\mathbf{I}_n$ based on $\mathbf{I}_{n-1}$:

$$\mathbf{I}_n = \eta\mathbf{I}_{n-1} + \mathbf{D}_n. \tag{5.6}$$

We use Equation 5.5 to characterize the drift between different system configurations. In our network training experiments (see Section 7.3), we observed that it is challenging to transfer the control parameters from a virtual trained example to a real fabricated UAV. Adding noise into the simulation model can only help marginally. For example, a controller that works for a simulated UAV in tracking a certain target velocity trajectory will exhibit a constant drift when tracking the same trajectory for a real UAV with slightly different physical configurations. This constant drift issue is named the *steady-state error* in control theory [5].

The key idea of this error integral is motivated by the classical PID controller. Recall that a traditional PID controller consists of three terms: P is proportional to the current tracking error, I is a integration of past error over time, and D is the derivative of error which is a estimate of the future trend of the system. The integral term I is designed to eliminate the system's steady-state error by consistently penalizing the accumulated drifts over time. Inspired by that, we append an error integral term to the state input of our neural network model to mimic the behavior of PID in minimizing the accumulated systematic error. Such drift is recorded in every time step and penalized as an energy term in the objective function (see Section 6) when training the neural network.

Chapter 6

Controller Training Method: Reinforcement Learning

6.1 Framework

We train our controller using a standard reinforcement learning framework. Our control problem is modeled by a Markov decision process problem $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, s_0, \gamma, H)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}(s'|s, a)$ is the transition function which is a distribution of next states s' given the current s and action a, $R(s, a)$ is a scalar reward the agent can get after taking action a from state s, s_0 is the start state, $\gamma \in [0, 1)$ is a discount factor, and H is the maximum horizon. A policy $\pi_\theta(a|s)$ models the probability of choosing action a in state s. The goal is to find the parameters θ of a policy $\pi_\theta(a|s)$ that maximizes the expected finite-horizon reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\big[\sum_{t=0}^{H} \gamma^t R_t\big], \tag{6.1}$$

where $p_\theta(\tau) = p(s_0) \prod_{t=0}^{H-1} \mathcal{P}(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$ is the probability distribution over all trajectories $\tau = (s_0, a_0, ..., s_H, a_H)$, and $p(s_0)$ is the initial state distribution.

In our work, the state s is a 13 dimensional vector defined in Equation 5.1, and the action a consists of N values representing the thrust of each rotor. The policy π

is our neural network controller. We use Proximal Policy Optimization [35] (PPO), the state-of-the-art policy gradient method, to train the controller.

6.2 Reward

Our reinforcement learning reward R at each step t consists of five terms – the velocity tracking error, energy efficiency, flight stability, error integral, and orientation. The philosophy of this reward function is to enable the UAV to match a prescribed trajectory in the velocity space and meanwhile to optimize other objectives of motion. We give the definition of the reward function for the current time instant as:

$$R(\mathbf{s}, \mathbf{a}) = d - w_v c_v(\mathbf{s}) - w_a c_a(\mathbf{a}) - w_\omega c_\omega(\mathbf{s}) - w_I c_I(\mathbf{s}) - w_\psi c_\psi(\mathbf{s}). \tag{6.2}$$

In Equation 6.2, d is a constant alive bonus, c_v is the velocity cost, c_a is the action cost, c_ω is the stability cost, c_I is the integral cost, and c_ψ is the orientation cost, and w_v, w_a, w_ω, w_I, w_ψ are their corresponding weights. The role of the alive bonus d is to keep the controller alive as long as possible by providing a positive reward at each time step.

The need for the first three terms stems directly from the definition of a velocity-tracking controller for hybrid UAVs. The velocity cost c_v penalizes the deviation of the current velocity from the target velocity as $c_v(\mathbf{s}) = \|v_t - \hat{v}\|_2^2$. The action cost $c_a(\mathbf{a})$ aims to maximize the leverage of the wing lift forces by penalizing the thrust output: $c_a(\mathbf{a}) = \|\mathbf{a}\|_2^2$ (see the usage of the wing lift in Figure 8-4). The stability cost $c_\omega(\mathbf{s})$ prevents the jittering motion by minimizing angular velocity: $c_\omega(\mathbf{s}) = \|\omega\|_2^2$.

The fourth term, the integral cost c_I, attempts to zero out the steady state error by discouraging large integral error: $c_I(\mathbf{s}) = \|I_u^2 + I_v^2 + I_w^2 + I_\psi^2\|_2^2$. As previously discussed, this term is essential for bridging the reality gap.

Finally, the orientation cost $c_\psi = D_\psi^2$ encourages the controller to fly along the desired orientation as straight as possible by penalizing large D_ψ. As our velocity is defined in a re-oriented coordinate system, a UAV that travels along an arbitrary curve with its heading consistent with the tangent direction and its velocity being of the same magnitude as $\hat{v}$ also has near-zero velocity cost.

The term c_ψ takes the yaw angle ψ at each time step into account to penalize those undesired turning motions. As the control of a UAV is affected not by the actual yaw value but by the yaw error, we always set $\hat{\psi}$ to be zero during training. We show the ablation test for the orientation cost in Table 7.4.

Chapter 7

Experiment and Evaluation in Simulation

In this chapter, we provide the details about the models for evaluation, training specifications and the evaluation of our algorithm in simulation. We train our controller for five hybrid UAV models in a physics simulator enhanced by a set of perturbation terms. We report the implementation details for the training process in Section 7.2. We evaluate our controller in Section 7.3 by conducting a series of ablation tests to demonstrate the efficacy of each reward term.

7.1 Hybrid UAV Models

To verify that our method is general for different UAV configurations, we developed a CAD interface within a parametric CAD tool [26] to design a variety of different models. We designed a number of subcomponents such as wings, fuselages, and propeller mounts. We define a handful of parameters for each components that control shape and size variations. Users can create new designs by mixing, matching and manipulating parameters of the components. Once the design is finished, system identification (Table 7.1) is measured and computed based on the assembly of its specific components. The same numbers are used in training and simulation. We created the following virtual hybrid UAV models (Figure 7-1) to test the algorithm,

model	quad-plane	tail-sitter	x-wing
mass	1.779	1.108	1.532
diagonal inertia x	0.194	0.13	0.184
diagonal inertia y	0.064	0.025	0.191
diagonal inertia z	0.253	0.149	0.336
wingspan	1.38	1.43	1.51
max thrust	7	7	7

Table 7.1: System identifications of the hybrid UAVs. Numbers are in SI units.

which we briefly describe below.

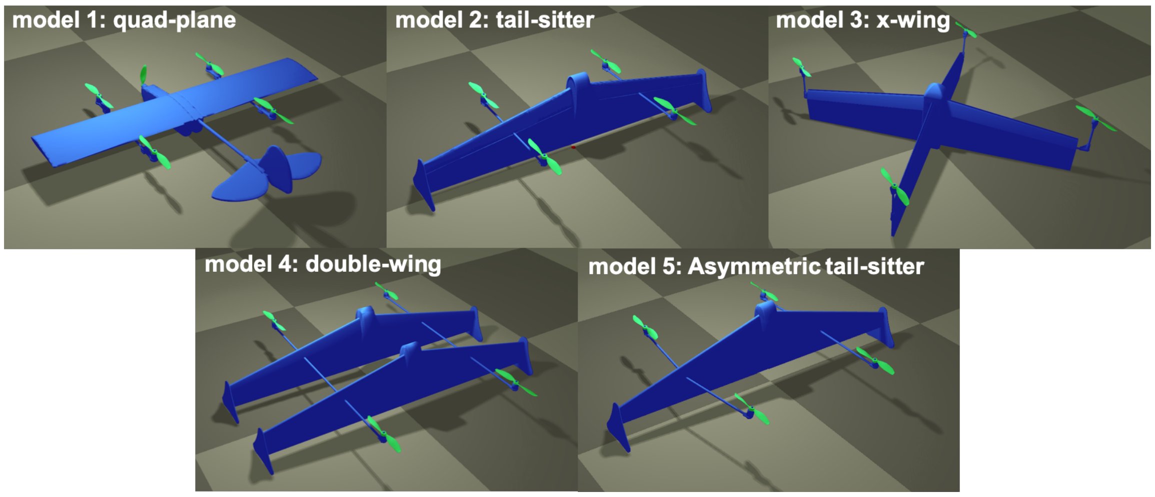

Figure 7-1: Four hybrid UAV models used in our training and evaluation process. Note that their dynamics are quite different, making it a non-trivial task to design a control scheme that works for all of them.

Quad-plane Our quad-plane is modeled based on the corresponding real plane we built. We use the system identification data for its physical parameters in simulation. The four vertical rotors are primarily used to cancel out gravity, and the front rotor is the major source of forward propulsion. For simplicity, we ignore the influence of the propellers on the aerodynamic models of the wings.

Tail-sitter The virtual tail-sitter model is equipped with a flat wing and two pairs of rotors. To enter the plane mode, the tail-sitter is tilted forward to allow the wings to generate lift. This transition is challenging because the UAV needs to change its pose aggressively in a timely fashion: as the tail-sitter leans forward, the vertical net thrust quickly decays while the wings may yet not be ready to generate enough lift to compensate. This can easily cause the UAV to lose altitude if the controller is slow to respond.

X-wing The X-wing model is a variant of the aforementioned tail-sitter, and therefore brings similar challenges when transitioning from hovering to gliding, even in simulation. This UAV consists of two orthogonal pairs of wings and four rotors at the end of each wing. Like a classic quadcopter, the design is symmetric around the axis of the fuselage, giving it good performance in copter mode.

Double-wing This example is another variant of a tail-sitter but with two pairs of parallel wings. We compute the lift and drag of each wing separately, assuming no interference in between. Rotors are evenly distributed on both sides and are connected by two rods orthogonal to the wings. Two fuselages are modeled between each pair of wings. Transitioning to the plane mode requires the fuselage to lean forward until the wings start to generate sufficient lift.

Asymmetric tail-sitter We design this non-traditional hybrid UAV models to test the robustness of our controller algorithm. Essentially, this is the same tail-sitter as the second example with randomly translated (within 0.1m) and rotated (up to 15 degrees) rotors.

Note that the dynamics of these models is significantly different: for example, in order to make use of the wings when advancing, the quad-plane model needs to simply turn on the front rotor and mildly adjust the other four rotors, while the tail-sitter model has to lean forward drastically so that the wings become horizontal.

7.2 Training

We implemented our algorithm in Python and C++ using the PPO algorithm in OpenAI Baselines [15].At the beginning of each trajectory, a target velocity and the initial state are randomly generated, and the UAV attempts to track this velocity in a simulator at 100Hz. Rewards are collected at each time step. The simulation is terminated after a fixed duration (20s in our experiments) or the UAV cannot maintain a stable flying status, after which the simulation restarts with a new target velocity and initial state. The epoch terminates when the total number of simulated time steps reaches 4096. We ran PPO for 30 million time steps using the same hyperparmeters in Table 7.2 for all models. Training each model took 70 minutes with 32 parallel MPI calls on a workstation of 112 CPU cores and 500G memory, and convergence was usually achieved well before the training ended (Figure 7-2).

During training, we added sensor noise, signal latency, and perturbed dynamic models to improve the robustness of the controller as suggested in previous work [37, 29]. We summarized their specifications in Table 7.3. In terms of perturbing the dynamic model, we further constrained the change of thrust at each time step to be below $0.4N$. As a result, the system had to learn to avoid generating unreasonably high frequency control signals.

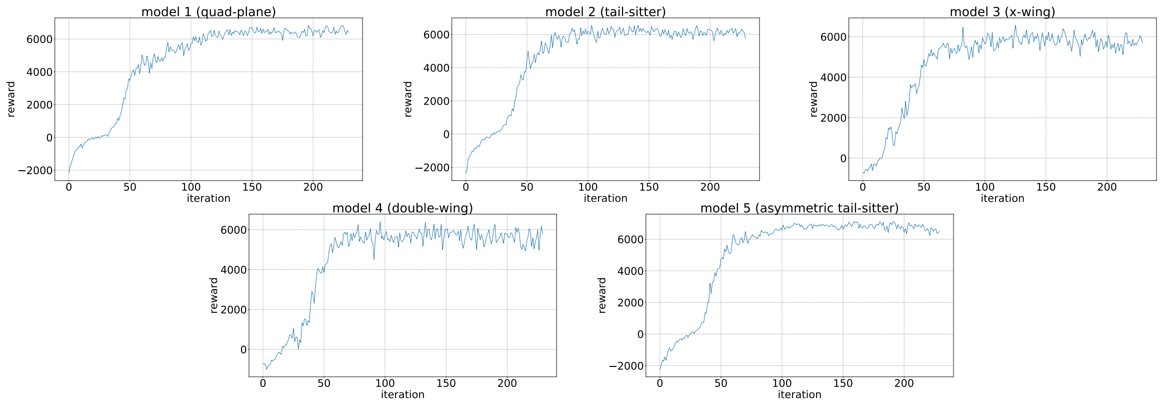

Figure 7-2: The training progress on 5 hybrid UAV models using 30 million timesteps in 70 minutes.

7.3 Evaluation

We conducted ablation study on all our five virtual models (Figure 7-1) in simulation to validate the necessity of the new components we introduced in the training pipeline. The ablation test shows that our unique reward definition and the integral block are

name	value	name	value
`timesteps_per_actorbatch`	4096	`schedule`	linear
`clip_param`	0.2	η	0.999
`entcoeff`	0	d	4
`optim_epochs`	10	w_v	0.5
`optim_stepsize`	3e-4	w_a	2e-3
`optim_batchsize`	64	w_ω	5e-2
`gamma`	0.995	w_I	6e-2
`lam`	0.95	w_ψ	1.5
`adam_epsilon`	1e-5		

Table 7.2: Hyperparameters used in training all models. Please refer to [15], Section 5, and Section 6 for the meaning of each parameter.

category	noise distribution	parameter range
sensor noise	Gaussian	$\lvert\mu\rvert \leq 0.2, 0.005 \leq \sigma \leq 0.05$
mass perturbation	uniform	$\leq 5\%$
inertia perturbation	uniform	$\leq 40\%$
lift and drag	uniform	10% to 20%
signal delay	Gaussian	$\mu = 0.04, \sigma = 0.01$
time step delay	Gaussian	$\mu = 0.01, \sigma = 0.005$

Table 7.3: A summary of noise we added in the training process. All magnitudes use SI units. Percentage numbers mean the noise is relative.

the core reasons for our success in narrowing the reality gap.

Evaluation Metrics To evaluate the performance of a given controller in simulation, we consider the following metrics: first, we randomly generated a target velocity and simulated the UAV for up to 20s at 400Hz, the control frequency on real UAV platforms. The simulation stopped early once the controller was incapable of maintaining a steady flight. We report the velocity tracking error and the survival time averaged in 50 repeated experiments. We refer to this as the `RandVel` task. Second, we designed two velocity trajectories by concatenating multiple one-dimensional constant speed tracking tasks: advancing, backing, ascending, descending, and side sliding at a constant rate. The UAV was instructed to finish the selected tasks in order. We refer to them as the `VelTraj 1` and `VelTraj 2` tasks respectively. In `VelTraj`

1, we designed a 50s velocity trajectory and commanded the UAV first to climb up at a constant rate, followed by advancing then landing. In `VelTraj 2`, the task is more complex, including advancing and back, side sliding and back, descending and ascending, and finally advancing at a high speed. We repeated the experiment 25 times for each task and report the average. We conducted all tasks in our simulation with noise modelling.

In the remaining section, we report the ablation study on the following components we introduced in the training pipeline: (1) the orientation cost in rewards, (2) the sensor noise, signal latency, and the dynamics noise in simulation, (3) the integral block in the neural network controller, and (4) the usage of velocity difference as the input to the controller. We remove each component from the pipeline, retrain the controller using the same hyperparameters, and analyze the performance of the controller using the evaluation metrics above. This gives us 5 training environments to compare in total. We report the performance in these tasks in Table 7.4.

The Orientation Cost As can be seen from Column 3 and 4 of Table 7.4, penalizing large orientation error lengthens the survival time in general. Among all the 15 experiments, including the orientation cost in rewards improved the flight time in 12 of them. Specifically, it brought significant performance boost for `VelTraj 2`, which is the most complex task among the three. Moreover, by comparing Column 4 and 5 with Column 6 and 7, we can see the orientation cost and the noise are the two most indispensable factors for increasing the flight time. Based on these observations, we conclude that by discouraging large heading angles, it is easier for the UAV to maintain a steady flight.

Compared with its influence on the flight time, the improvement on velocity tracking is less notable, part of the reason being that the orientation cost does not consist of any velocity quantities.

Sensor Noise, Latency, and Perturbation on Dynamics Column 5 of Table 7.4 summarizes the performance of the controller after disabling all types of noise mentioned above. Together with Column 3, Column 5 shows that disabling noise in training has very strong negative effects on both metrics. In fact, after excluding noise, the performance became the worst in 10 and 5 out of the 15 experiments on flight time and velocity tracking error respectively.

Another interesting observation is that disabling noise is the least effective on model 1 among all models, especially for the flight time. Noting that the other three models are all variants of tail-sitters, this implies the quad-plane dynamic model is more resistant to noise. This is not surprising: a tail-sitter needs to change its pitch aggressively to switch between plane and copter modes, and therefore even a moderate perturbation may easily flip it over.

Integral Block Quantitative results on the performance of the integral block are reported in Column 6 of Table 7.4. Comparing Column 6 with Column 3, we observe that adding the integral block suppressed the velocity residual moderately in 12 out of the 15 experiments. This is consistent with the behavior of an integral controller in the classic PID control theory, i.e., it can zero out the steady-state error given enough time. The velocity tracking errors in our experiments did not end up being zero because of the noisy environment as well as the fact that we were chasing a changeable target velocity in `VelTraj` tasks.

We also notice the improvement of the survival time after introducing the integral block, but it is less significant. As a result, it is a good complement to the aforementioned orientation cost, which increases the flight time by a large margin but is less effective in reducing velocity tracking errors.

The combination of the orientation cost, the integral block, and the noise and latency model is the core reason for successfully bridging the simulation to reality gap when we transfer the trained controller to hardware, as shown in our video. Furthermore, the performance drop between Column3 and Column 4 to 6 indicates that all three components are indispensable.

Velocity Inputs In this experiment, we fed the desired and actual velocity as opposed to their difference into the neural network controller and report the results in Column 7 of Table 7.4. Among all the 30 experiments, the resulting controller outperformed ours in 9 experiments and was dominated by ours in fifteen, with the remaining six being the same. This modification to the velocity inputs is essentially nothing but a simple arithmetic operation. Given that our neural network has two hidden layers of 64 neurons, the network may easily absorb this operation into its control policy, which explains why this modification results in a controller with comparable performance to ours. Nevertheless, we chose to use our original design in all experiments because it led to a more compact input representation.

metric	model	ours	w/o c_ψ	w/o noise	w/o integral	w/o vel diff
`RandVel`	1	**0.35**	0.45	0.49	0.42	0.51
velocity	2	0.44	0.40	0.39	0.40	**0.37**
tracking	3	**0.45**	0.51	0.51	0.52	0.51
error (m/s)	4	**0.41**	0.57	0.50	0.57	0.43
	5	**0.34**	1.14	0.49	0.56	0.46
`RandVel`	1	**20.00**	19.98	**20.00**	19.61	19.98
survival	2	**20.00**	**20.00**	11.67	19.62	19.95
time (s)	3	**20.00**	19.90	14.25	**20.00**	**20.00**
(max: 20s)	4	17.71	19.24	13.13	19.26	**20.00**
	5	**20.00**	14.64	11.66	19.66	**20.00**
`VelTraj 1`	1	**0.47**	0.59	0.64	0.55	0.61
velocity	2	0.57	0.73	0.81	0.61	**0.53**
tracking	3	0.73	0.69	0.96	0.78	**0.66**
error (m/s)	4	0.70	0.84	0.93	0.89	**0.63**
	5	0.62	0.60	**0.57**	0.58	0.63
`VelTraj 1`	1	**50.00**	**50.00**	49.07	**50.00**	**50.00**
survival	2	**50.00**	47.54	10.55	**50.00**	47.35
time (s)	3	**50.00**	14.76	10.44	**50.00**	**50.00**
(max: 50s)	4	40.66	14.12	10.90	34.43	**50.00**
	5	44.83	10.73	10.60	43.81	**50.00**
`VelTraj 2`	1	**0.39**	0.95	0.56	0.65	0.58
velocity	2	0.43	0.59	0.45	**0.41**	0.45
tracking	3	**0.53**	0.66	0.57	0.69	0.55
error (m/s)	4	0.53	0.53	**0.49**	0.60	0.67
	5	**0.44**	0.61	0.48	0.61	0.52
`VelTraj 2`	1	**100.00**	39.71	93.33	**100.00**	**100.00**
survival	2	**100.00**	71.20	69.63	**100.00**	97.69
time (s)	3	**100.00**	65.02	70.80	73.78	**100.00**
(max: 100s)	4	89.05	47.87	65.30	76.87	**97.70**
	5	92.57	63.61	70.60	91.25	**100.00**

Table 7.4: The average velocity tracking error (smaller is better) and survival time (larger is better) of different controllers in the task of tracking random velocities and two trajectories. Bold number is the best in each row. The meaning of each column is as follows: ours: enabling all features; w/o c_ψ: training without the orientation cost in the reward function; w/o noise: training without sensor noise, signal latency, or perturbation in the dynamic model; w/o integral: training without the integral block from the neural network controller and the integral term in the reward function; w/o vel diff: using the target and sensed velocities instead of their difference as the input to the neural network in training.

Chapter 8

Fabrication and Flight Tests

We fabricate three UAVs using the design models (including the quad-plane, the tail-sitter, and the x-wing, see Figure 8-1) discussed in the previous section. First, we will discuss the fabrication process in Section 8.1, and then we will present a set of flying tests and log analyses in Section 8.2 to demonstrate the aerodynamic performance of our hybrid UAVs.

8.1 Fabrication

After the shape design is fixed and the controller is finalized in simulation, we generate a corresponding fabrication plan. We fabricated three UAV models to verify our trained controller in real flight. For the hybrid UAVs, fabrication processes include 3D printing, laser cutting and manual assembly. Structural stiffness and weight optimization are paramount to ensure the flight performance of the hybrid UAV. During the materials selection process, we optimize the design to limit weight and maximize structural stiffness. To ensure structural stiffness, each hybrid UAV has an internal chassis constructed from carbon fiber tubing and fiberglass reinforced 3D printed nylon connectors. To construct the wings, we used two different architectures: one built from laser cut foam, and one built from a balsa wood frame and heat shrink plastic film. Laser cut foam wing are more durable, while the heat shrunk wings allow for more effective lift generation. The designer can choose from one of these techniques

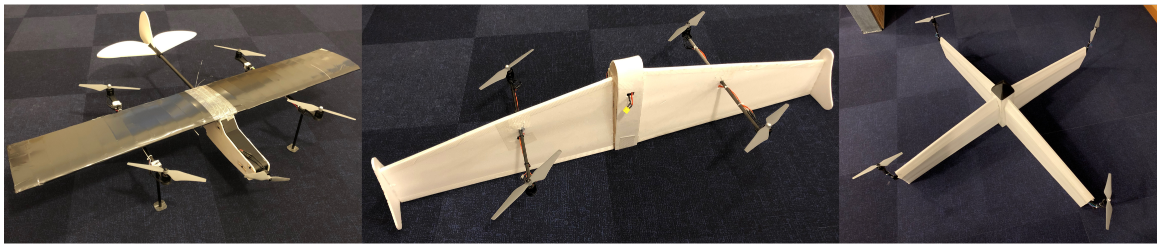

Figure 8-1: The fabricated hybrid UAV models. From left to right: quad-plane (model 1), tail-sitter (model 2), x-wing (model 3).

depending on preferred performance.

The flight controller runs our modified version of the open source software ArduCopter [4] on a Pixhawk flight computer hardware [25]. The hybrid UAVs' propellers are driven by brushless electric motors.

We briefly describe the three fabricated UAVs below:

Model 1: Quad-plane Our custom-built quad-plane (Figure 8-1 left) is based on a classic fixed-wing plane model plus overlapping the rotor configuration of a standard quadcopter. We first built the fuselage, the wing, and the tail and assembled them into a fixed-wing model. We then added four rotors on both sides of the wing symmetrically. The wings and the tail did not have articulated control surfaces, so they generated lift and drag passively. The attitude of the UAV was primarily controlled by the net effects of all five rotors.

Model 2: Tail-sitter The tail-sitter model (Figure 8-1 middle) is a mixture of a delta-wing plane and a quadcopter. We first assembled the fuselage and the pair of wings, after which we placed two rods with rotors on both ends in a position perpendicular to the wings and equally distant to the body. We added two wingtips so that the tail-sitter could take off from a vertical position on land.

Model 3: X-wing The x-wing plane (Figure 8-1 right) consists of two pairs of wings perpendicular to each other. All wings were attached to the four sides of the central fuselage. We placed four rotors at the end of each wing and intentionally lifted them a little higher to reduce the interference between propellers and wings.

8.2 Flight Tests

During the flight test, the UAV first loitered for a few seconds to show that the controller can stabilize the body in the air, which is the key feature of a multicopter. We then instructed the UAV to advance directly from the loitering state, exhibiting the aerodynamic behavior of a fixed-wing plane. In contrast to other hybrid UAV controllers, which typically execute a manually designed mode transition from hovering to gliding, our neural network controller does not store any modes and can switch modes implicitly by simply updating target velocities. The UAV then pulled itself back to loiter and land. To demonstrate the generality of our approach, we perform the same flight tasks on three hybrid UAVs which have dramatically different dynamics. We highlight the transition between hovering and advancing for all three models in Figure 8-2 and Figure 8-3. Note that for each model, the same controller was used both in simulation and in the flight video. We ask readers to refer to our video for the full flight tests.

Our flight logs (Figure 8-4) recorded the output thrust of each rotor over time during the flight, labeled by different colored curves. We computed the vertical component of the net thrust and plotted it as a gray curve in the log. To assist the analysis, we also plotted the net external force excluding gravity in the vertical direction in orange, which we estimated from the motion data. Assuming the UAV maintained its altitude all the time during the flight, we can use the margin between the orange and gray curves to estimate the lift from the wings. Finally, we highlighted the hovering, transition, and advancing using different background colors.

Model 1: Quad-plane Traditionally, the front rotor of the quad-plane is solely used in the plane flight mode. Our flight log (Figure 8-4, top) shows that it also contributed to resisting gravity when the UAV was loitering. This novel usage of the front rotor came from the fact that the neural network controller was trained to intentionally fuse the copter and plane flight modes. We also notice that the wings started to provide lift as soon as the UAV entered its advancing mode (as expected), as the net thrust curve consistently lies below the reference weight line.

Model 2: Tail-sitter During the loitering mode, the tail-sitter behaved quite stably as indicated by the almost constant output thrust from each rotor (Figure 8-4, middle). The transition from loitering to advancing is significantly harder than the quad-plane because it needs to lean forward by a large amount before the wings become active. Still, our neural network controller was capable of transitioning smoothly from loitering to advancing. From the flight log, we can see that both the rotors and wings were active when the vehicle was cruising afterwards, making the flight more efficient than a pure multicopter.

Model 3: X-wing The X-wing shares quite a few similarities with the previous tail-sitter model, e.g., steady loitering mode and challenging transition. After entering the cruising mode, the margin between the vertical net force and the vertical net thrust is particularly large because it has the longest wingspan of the three models, making it possible to generate powerful lift.

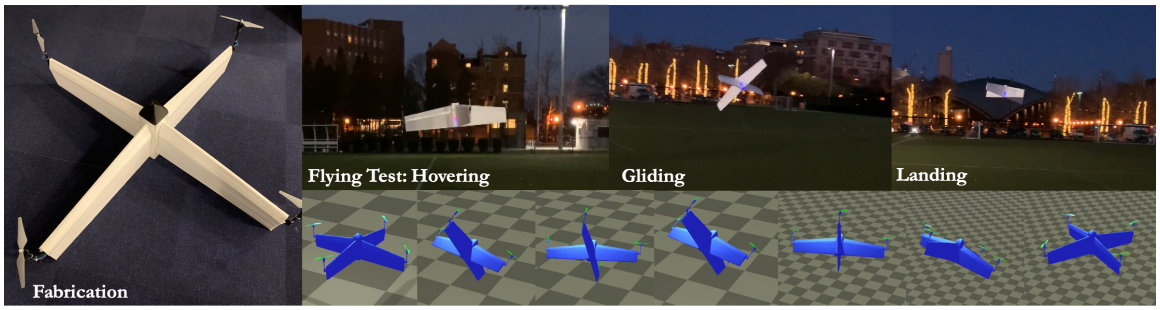

Figure 8-2: Flight tests for X-Wing. The fabricated UAV is shown on the left, and the real flight tests and virtual flight tests are shown on the right.

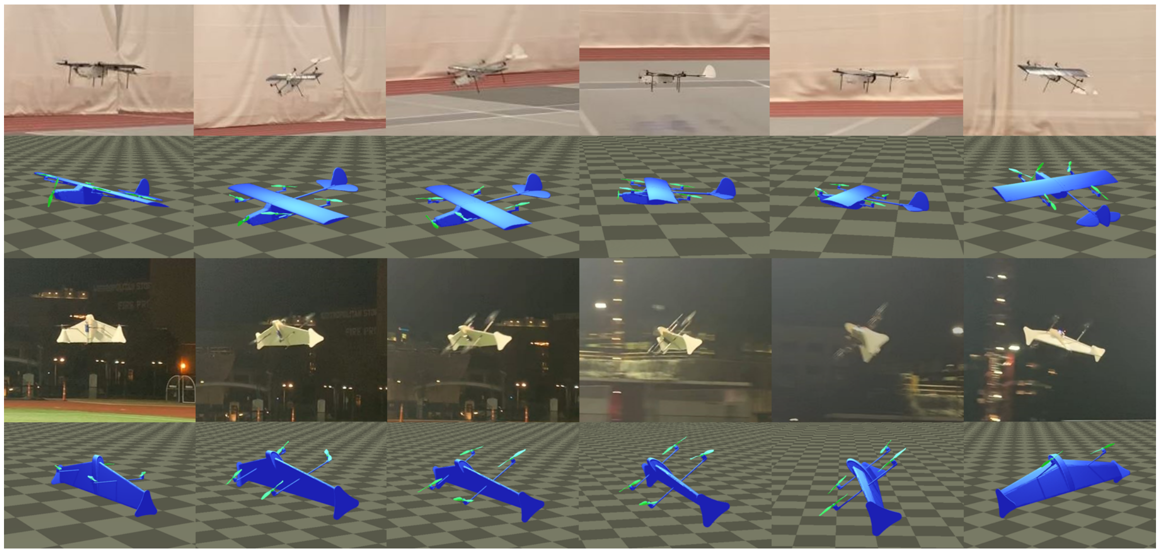

Figure 8-3: The quad-plane (model 1) and the tail-sitter (model 2) are transitioning from loitering to gliding (row 1 and 3). Their corresponding flight simulation are shown below (row 2 and 4).

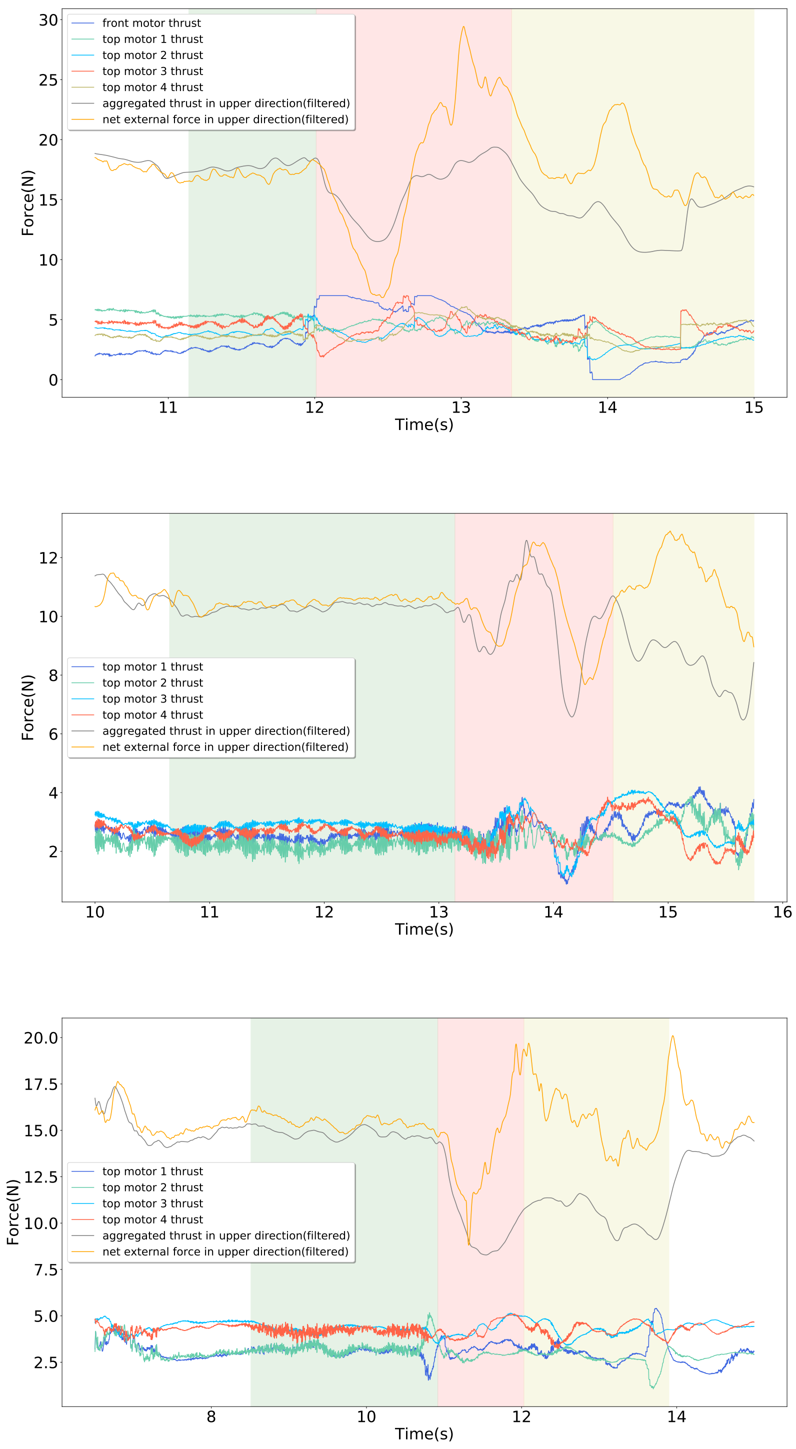

Figure 8-4: The flight logs of the quad-plane (top), tail-sitter (middle) and X-wing (bottom). The shaded background represents the time interval when the UAV was loitering (green), transitioning (red), and starting to advance (yellow). The colored curves other than the orange ones correspond to the thrust from each rotor. The gray curve shows the vertical component of the net thrusts, and the orange curve presents the estimated net external force excluding gravity of the UAV. The difference between the orange and gray lines indicates the lift the wings contributed.

Chapter 9

Discussion and Limitations

In this chapter, we make the conclusion of our work and discuss about the limitations and potential works.

The simulation experiments show that our *model-agnostic* method is able to control widely different configurations of hybrid UAVs and our *mode-free* controller enables flight in all six directions from an input velocity. The ablation tests show that there is a strong motivation for the components we introduce in the pipeline. Most importantly, borrowed from the classic PID control theory, the *integral block* decreases the velocity tracking error. Finally, our flight tests show how our proposed solution can bridge the reality gap.

Our method still leaves room for further developments. First, the analytic flat-plane model does not take into account complex aerodynamic effects between the airflow and the fuselage. We have chosen this approach because it allows real-time simulation in the training process. While such model is qualitatively good enough for building real fixed-wing plane models [8], it would be interesting to investigate if a more accurate simulation could lead to better results in the case of hybrid UAVs.

Second, our proposed velocity tracking controller has no direct way to measure position, and therefore the deviation between a desired flight path and the actual trajectory of the hybrid UAV could be large. In the future, it would be valuable to add position sensors and design a trajectory tracking controller.

Third, it would be interesting to increase the maneuverability of the vehicles.

While the UAVs can efficiently change pitch in order to maximize velocity, changes in the heading direction are slower because our method trains the copter to advance by keeping yaw at zero. Further training would be necessary to allow sharp turns.

Fourth, we restricted the scope of the problem to wings as passive actuators. Future work should investigate models with control surfaces (e.g., rudders) on the wings and tails.

Finally, it would be interesting to explore the parametric component database to develop algorithms for automatic design optimization based on flying capabilities, as in previous work [16]. In theory, one could couple the proposed automatic method for computing controllers with simulation to measure flight performance for a given input geometry. While this would make performance evaluation quite slow, it could be coupled with pre-computation techniques for interactive design exploration [36]. A more interesting approach would be to develop *concurrent* optimization methods directly in the reinforcement learning pipeline. While some approaches have been suggested [33], further study is still required in this direction.

Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, Nov. 2010.

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS '07, pages 1–8, 2007.

[3] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation, 2018.

[4] ArduPilot. Ardupilot open source autopilot, 2016.

[5] Karl Johan Aström and Richard M. Murray. *Feedback Systems: an Introduction for Scientists and Engineers.* Princeton University Press, 2010.

[6] Somil Bansal, Anayo K. Akametalu, Frank J. Jiang, Forrest Laine, and Claire J. Tomlin. Learning quadrotor dynamics using neural network for flight control. In *Proceedings of 2016 IEEE 55th Conference on Decision and Control*, CDC '16, pages 4653–4660, 2016.

[7] Roman Bapst, Robin Ritz, Lorenz Meier, and Marc Pollefeys. Design and implementation of an unmanned tail-sitter. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS '15, pages 1885–1890. IEEE, 2015.

[8] Andrew Barry. *High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2016.

[9] Gaurav Bharaj, David I. W. Levin, James Tompkin, Yun Fei, Hanspeter Pfister, Wojciech Matusik, and Changxi Zheng. Computational design of metallophone contact sounds. *ACM Trans. Graph.*, 34(6):223:1–223:13, October 2015.

[10] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 144–151, New York, NY, USA, 2008. ACM.

[11] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Trans. Graph.*, 32(4):83:1–83:12, July 2013.

[12] Rick Cory and Russ Tedrake. Experiments in fixed-wing uav perching. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7256, 2008.

[13] Ruta Desai, Ye Yuan, and Stelian Coros. Computational abstractions for interactive design of robotic devices. In *2017 IEEE International Conference on Robotics and Automation*, ICRA '17, pages 1196–1203, 2017.

[14] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation*, ICRA '17, pages 2169–2176, 2017.

[15] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines, 2017.

[16] Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. Computational multicopter design. *ACM Trans. Graph.*, 35(6):227:1–227:10, November 2016.

[17] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS '16, pages 4019–4026. IEEE, 2016.

[18] Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Trans. Graph.*, 37(4):160:1–160:12, July 2018.

[19] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[20] R. Hugh Stone, Peter Anderson, Colin Hutchison, Allen Tsai, Peter Gibbens, and K C. Wong. Flight testing of the t-wing tail-sitter unmanned air vehicle. *Journal of Aircraft - J AIRCRAFT*, 45:673–685, Mar. 2008.

[21] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, Oct. 2017.

[22] H Jin Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, NIPS '04, pages 799–806, 2004.

[23] Libin Liu and Jessica Hodgins. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Trans. Graph.*, 37(4):142:1–142:14, July 2018.

[24] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3d-printable robotic creatures. *ACM Trans. Graph.*, 34(6):216:1–216:9, October 2015.

[25] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1-2):21–39, 2012.

[26] OnShape. https://www.onshape.com/, 2019.

[27] Atsushi Oosedo, Satoko Abiko, Atsushi Konno, Takuya Koizumi, Tatuya Furui, and Masaru Uchiyama. Development of a quad rotor tail-sitter vtol uav without control surfaces and experimental verification. In *2013 IEEE International Conference on Robotics and Automation*, ICRA '13, pages 317–322, May 2013.

[28] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018.

[29] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation*, ICRA '18, pages 1–8, May 2018.

[30] Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 36(4):41:1–41:13, July 2017.

[31] Robin Ritz and Raffaello D'Andrea. A global controller for flying wing tailsitter vehicles. In *2017 IEEE International Conference on Robotics and Automation*, ICRA '17, pages 2731–2738, May 2017.

[32] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[33] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. Jointly learning to construct and control agents using deep reinforcement learning. *arXiv preprint arXiv:1801.01432*, 2018.

[34] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, ICML '15, pages 1889–1897, 2015.

[35] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. Interactive design space exploration and optimization for cad models. *ACM Trans. Graph.*, 36(4):157:1–157:14, July 2017.

[37] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[38] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS '17, pages 23–30. IEEE, 2017.

[39] Nobuyuki Umetani, Takeo Igarashi, and Niloy J Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4):86–1, 2012.

[40] Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph.*, 33(4):65:1–65:10, July 2014.

[41] Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. How to train your dragon: Example-guided control of flapping flight. *ACM Trans. Graph.*, 36(6):198:1–198:13, November 2017.

[42] Jungdam Won, Jungnam Park, and Jehee Lee. Aerobatics control of flying creatures via self-regulated learning. *ACM Trans. Graph.*, 37(6):181:1–181:10, December 2018.